

HEWLETT  PACKARD

HP

ALGOL
reference manual

HP 2000 COMPUTER SYSTEMS

ALGOL

reference manual



HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

MANUAL PART NO. 02116-9072
MICROFICHE PART NO. 02116-91785

Printed: OCT 1974
Printed in U.S.A.

List of Effective Pages

Pages	Effective Date
Title	Oct 1974
ii	Oct 1974
iii through xiv	Apr 1974
1-1 through 1-21	Apr 1974
2-1 through 2-6	Apr 1974
3-1 through 3-23	Apr 1974
4-1 through 4-2	Apr 1974
5-1 through 5-3	Apr 1974
A-1 through A-3	Apr 1974
B-1 through B-4	Apr 1974
C-1 through C-9	Apr 1974
I-1 through I-2	Apr 1974

PREFACE

This book is the programmer's reference for the HP ALGOL language for the Hewlett-Packard 2000 series computer systems.

HP ALGOL is similar to the source language defined by the ALGOL 60 Revised Report, *Communications of the ACM*, January, 1963. The reader should be familiar with ALGOL 60 and with the Hewlett-Packard computer and software systems, either through completion of Hewlett-Packard training courses or through equivalent experience. The publication is a programmer's reference manual and is not intended to be used as a training manual.

This edition of *HP ALGOL Programmer's Reference Manual* supersedes all earlier editions and incorporates all Manual Change Sheets issued to date.

Special features and limitations peculiar to HP ALGOL are described in the Introduction. Section I describes the structure of ALGOL, defining declaration, expressions, and statements. Procedures are described in Section II. Section III covers INPUT/OUTPUT and READ/WRITE operations and the use of FORMAT specifications. Section IV explains the use of the ALGOL compiler. Section V describes how to use procedures from FORTRAN or Assembly Language procedures in an ALGOL program. Error messages and program examples are contained in the Appendix Section.

Since Hewlett-Packard provides compilers for several operating systems, consult the appropriate manuals below for operating procedures and changes local to each system:

- Basic Control System* (02116-9017)
- Disc Operating System* (02116-91748)
- DOS-III Disc Operating System* (02100-90136)
- Magnetic Tape System* (02116-91752)
- Moving-Head Disc Operating System* (02116-91779)
- Real-Time Software* (02116-9139)
- SIO Subsystem Operation* (5951-1390)
- Software Operating Procedures*

For details on the Formatter, consult the *Relocatable Subroutines* manual (02116-91780).

CONTENTS

iii	PREFACE
ix	INTRODUCTION
1-1	SECTION I PROGRAM STRUCTURE
1-1	DECLARATIONS
1-1	TYPE
1-2	ARRAY
1-3	LABEL
1-3	SWITCH
1-4	EQUATE
1-5	BLOCKS
1-6	ARITHMETIC EXPRESSIONS
1-6	Operators and Types
1-8	Conditional Expressions
1-9	VARIABLES
1-9	Simple Variables
1-9	Subscripted Variables
1-10	ASSIGNMENT STATEMENTS
1-10	Types
1-11	GO TO STATEMENTS
1-12	Designational Expressions
1-12	Labels
1-12	CONDITIONAL STATEMENTS
1-14	BOOLEAN EXPRESSION
1-15	Boolean Variables
1-16	WHILE STATEMENTS
1-16	DO STATEMENT
1-17	COMPOUND STATEMENTS
1-18	FOR STATEMENTS
1-20	CASE STATEMENT
1-21	PAUSE STATEMENT

2-1	SECTION II PROCEDURES
2-1	PROCEDURE DECLARATIONS
2-3	PROCEDURE STATEMENTS
2-3	CALL BY NAME/VALUE
2-5	FUNCTION PROCEDURES
2-6	CODE PROCEDURES
2-6	INTRINSIC PROCEDURES
3-1	SECTION III INPUT/OUTPUT
3-1	LIST DECLARATIONS
3-2	FORMAT DECLARATIONS
3-2	FORMAT SPECIFICATIONS
3-3	Ew.d Output
3-4	Ew.d Input
3-8	Fw.d Output
3-9	Fw.d Input
3-9	Iw
3-10	Aw
3-11	@w and Kw
3-12	nX
3-13	$nHh_1h_2\dots h_n$
3-15	$r"h_1h_2\dots h_n"$
3-15	NEW RECORD
3-16	REPEAT SPECIFICATIONS
3-18	FREE FIELD INPUT
3-18	Data Item Delimiters
3-19	Floating Point Input
3-20	Octal Input
3-20	Record Terminator
3-20	List Terminator
3-21	Comments
3-21	READ AND WRITE STATEMENTS
3-22	In-Line FORMAT Declarations
3-22	MAGNETIC TAPE STATEMENTS
3-23	UNIT-NUMBER

4-1	SECTION IV USING THE HP ALGOL COMPILER
4-1	CONTROL STATEMENT
4-2	OPERATING PROCEDURES
5-1	SECTION V FORTRAN AND ASSEMBLER SUBROUTINES
5-1	CALLING FORTRAN SUBROUTINES FROM ALGOL
5-1	CALLING ALGOL SUBROUTINES FROM FORTRAN
5-2	CALLING ALGOL PROCEDURES FROM ASSEMBLY LANGUAGE
5-2	CALLING ASSEMBLY LANGUAGE PROCEDURES FROM ALGOL
A-1	APPENDIX A — ALGOL ERROR MESSAGES
B-1	APPENDIX B — RUN-TIME ERROR MESSAGES
C-1	APPENDIX C — EXAMPLES
	INDEX
	TABLES
xi	Table I-1. Reserved Identifiers

List of Effective Pages

Pages	Effective Date
Title	Oct 1974
ii	Oct 1974
iii through xiv	Apr 1974
1-1 through 1-21	Apr 1974
2-1 through 2-6	Apr 1974
3-1 through 3-23	Apr 1974
4-1 through 4-2	Apr 1974
5-1 through 5-3	Apr 1974
A-1 through A-3	Apr 1974
B-1 through B-4	Apr 1974
C-1 through C-9	Apr 1974
I-1 through I-2	Apr 1974

INTRODUCTION

CHARACTER SET

HP ALGOL uses the following basic symbols:

Letters A through Z (upper case only)

Digits 0 through 9

Special Characters

+ - * / ↑ \ @ . : " ' ; () [] < > = # , \$ blank

Note the character differences between ALGOL 60 and HP ALGOL:

<u>ALGOL 60</u>	<u>HP ALGOL</u>
Letters (upper and lower case)	Only upper-case letters allowed
x	*
÷	\
$\leq$	<=
$\geq$	>=
≠	#
¬	NOT
∧	AND
∨	OR
⇒	(not allowed)
10	,
:=	← or :=
'...'	"..."

FUNDAMENTALS OF HP ALGOL

The basic symbols defined in the character set are the only symbols recognized by HP ALGOL. These symbols are combined to form identifiers, constants, variables, and specifications; these elements are then combined to form declarations and statements. Declarations and statements are combined into blocks and blocks into a program. Under the syntactic rules of ALGOL, a program can be a single block or several blocks, nested one within another. Similarly, a block can be a single statement or many statements separated by delimiters.

INTRODUCTION

IDENTIFIERS AND RESERVED IDENTIFIERS

An identifier is a name used by the programmer to identify a value, usually a variable. The first character of an identifier must be a letter; the succeeding characters may be letters or digits. Spaces may not appear within an identifier. The ALGOL compiler will recognize up to 15 characters; additional characters will be ignored. An identifier must be declared before it is used. (See Section I.)

Certain identifiers have been defined by the system as having specific meanings or functions. These "reserved identifiers" must not be used by the programmer to define his own variables. (See Table I-1.)

Table I-1
Reserved Identifiers

*ABS	END	LABEL	*SIGN	WHILE
AND	ENDFILE	*LN	SPACE	WRITE
*ARCTAN	*ENTIER	NOT	*SQRT	
ARRAY	EQUATE	OR	STEP	
BACKSPACE	*EXP	OUTPUT	SWITCH	
BEGIN	*FALSE	PAUSE	*TAN	
BOOLEAN	FOR	*PI	*TANH	
CASE	FORMAT	PROCEDURE	THEN	
CODE	GO	READ	TO	
COMMENT	IF	REAL	*TRUE	
*COS	INPUT	REWIND	UNLOAD	
DO	INTEGER	*ROTATE	UNTIL	
ELSE	*KEYS	*SIN	VALUE	

*These identifiers have been predeclared, in that the programmer can use them without declaring them. He may, however, override these declarations with his own.

INTRODUCTION

CONSTANTS

The HP ALGOL compiler recognizes four types of constant--decimal, octal, Boolean, and ASCII.

Decimal Constants

A decimal constant consists of the ten digits 0 through 9. If a decimal constant has no decimal point and no scale factor, it is of type INTEGER; a decimal constant with a decimal point and/or a scale factor is of type REAL. Either may have a sign. The apostrophe (') precedes the scale factor.

$$+3.1416'2 = +3.1416 \times 10^2 = +314.16$$

In the examples below, the first two constants are INTEGER; all the rest are REAL:

<u>ALGOL</u>	<u>NUMERIC VALUE</u>
0	0
177	177
.5384	.5384
+0.7300	.7300
-200.084	-200.084
+ 07.43'8	7.43×10^8
9.34'+10	9.43×10^{10}
2'-4	2×10^{-4}
-.083'-02	-.00083

ASCII Constants

An ASCII constant may contain one or two ASCII characters enclosed in quotation marks and is represented as follows:

"HP"

"A"

INTRODUCTION

If only one character appears between the quotation marks, it is interpreted as a null character followed by the character within quotes.

<u>ALGOL</u>	<u>INTERNAL REPRESENTATION</u>
"HP"	0100100001010000  H P
"A"	0000000001000001  null A

Octal Constants

An octal integer is represented by the character @ followed by a string of octal digits. No sign is allowed.

Examples: @123

@100

Note that the constants "A", @101, and 65 are all equivalent.

Boolean Constants

There are two Boolean constants -- TRUE and FALSE.

COMMENTS

Comments may be inserted in an ALGOL program for clarity. A comment is indicated either by the word COMMENT or by the ampersand (&).

The word COMMENT may be inserted anywhere in the program; all symbols between the word COMMENT and the next semicolon are treated by the compiler as comments and are printed in the listing, but do not appear in the object code. Comments of this type may be continued for many lines.

INTRODUCTION

The (&) ampersand symbol is used for short comments. All symbols to the right of the ampersand are treated as comments in the line in which the & symbol appears. The next line of the program is treated as normal coding unless the & symbol is repeated. The compiler also ignores these comments when generating object code but prints them on the listing. Following an END symbol, all symbols up to the next END, ELSE, or semicolon are treated as comments.

In the following example the third line, following the symbol BEGIN, and all of the fourth line are comments. The last line is also treated as comments.

```
REAL PROCEDURE INNERPRODUCT (A,B,N);  
VALUE N; INTEGER N; ARRAY A, B;  
BEGIN COMMENT INNERPRODUCT COMPUTES  
    SUM FROM I = 1. ..N OF A[I] *B[I];  
    REAL S; INTEGER I;  
    S ← 0;  
    FOR I ← 1 STEP 1 UNTIL N DO S ← A[I] *B[I] +S;  
    INNERPRODUCT ← S  
END OF INNERPRODUCT
```

SECTION I

PROGRAM STRUCTURE

An ALGOL program is made up of two kinds of elements -- declarations and statements. Declarations define the variables used in the program and state their properties. Statements have the character of a command or order.

A program has the form:

```
BEGIN D; D; . . . D; S; S; . . . S; S END$
```

The semicolon (;) separates one unit from the next; blank spaces are significant only in certain FORMAT specifications and must not be used in identifiers. Since END is not a statement, no semicolon is necessary after the final statement.

The \$ informs the compiler that this is the end of the program.

DECLARATIONS

Declarations describe the properties of identifiers. HP ALGOL accepts all declarations defined for ALGOL 60 except OWN. (HP ALGOL treats all variables as OWN.) HP ALGOL also accepts an EQUATE declaration.

TYPE

Type declarations declare that certain identifiers represent simple variables of a given type. There are three different types--integer, real, and Boolean. A type declaration consists of the symbol specifying the type, followed by a list of identifiers separated by commas.

EXAMPLES:

```
INTEGER I, J, K;  
REAL    X, Y, Z;  
BOOLEAN FLAG;
```

PROGRAM STRUCTURE

Dynamic allocation of storage does not occur in HP ALGOL. ALGOL 60 OWN declaration is unnecessary, since all variables are treated as OWN and declared variables are initialized by the compiler. Initialization of a variable is specified by following the variable in the declaration by an assignment symbol and a constant.

EXAMPLES:

```
INTEGER  I, J ← 10, K ← 0;
REAL     X ← 1.5,   Y ← -2'5, Z;
BOOLEAN  FLAG ← FALSE;
```

ARRAY

An ARRAY declaration declares that one or more identifiers represent multi-dimensional arrays, giving the number of dimensions and the lower and upper bounds of each dimension. The general form is:

```
type ARRAY  array name, ..., array name [bounds, ..., bounds], ...,
            array name, ..., array name [bounds, ..., bounds]
```

EXAMPLES:

```
REAL ARRAY  A [1:100];
INTEGER ARRAY I, J [1:10, 1:10], K [-50:50];
ARRAY  A, B, C [0:10];
```

An array may be REAL, INTEGER, or BOOLEAN. If no type is given, REAL is assumed.

The subscript bounds for each array are given in the first pair of brackets following the identifier of this array in the form of a bound pair list. Each item of this list gives the lower and upper bounds of a subscript; the two bounds are separated by a colon (:). Since storage allocation is not dynamic, the bounds are restricted to integers. Each lower bound must not be greater than the corresponding upper bound.

PROGRAM STRUCTURE

Array elements are stored by rows in ascending order of storage locations. For example, integer array A[3,3] has its elements stored as A[1,1], A[1,2], A[1,3], A[2,1], and so forth.

An array may have its elements initialized within the declaration. This is accomplished as in the following examples:

```
INTEGER ARRAY DIGITS [0:20] ← "0", "1", "2", "3", "4",  
                                "5", "6", "7", "8", "9";
```

DIGITS [I] is initialized to the ASCII equivalent of I, for $I \leq 9$.

DIGITS [I] is initially undefined, for $I > 9$. In a single array declaration, only the last array named may be initialized. Other arrays are initialized in other declarations.

LABEL

The LABEL declaration specifies the identifiers of statements to which control may be passed by a GO TO statement.

EXAMPLE:

```
LABEL AL, L, M1, SAM;
```

SWITCH

A switch is a set of labels which can be entered as objects of a GO TO statement.

```
SWITCH <switch identifier>← <label> ,..., <label>
```

The labels of a SWITCH declaration are associated from left to right with the set of positive integers. When the switch identifier is used as a subscripted expression in a GO TO statement, the label associated with the integer value of the subscript becomes the object of the GO TO statement.

PROGRAM STRUCTURE

EXAMPLE:

```
LABEL  L1, L2, L3, L4;  
SWITCH S ← L1, L2, L3, L4;  
      ⋮  
I←3;  
GO TO S[I];
```

Since the value of the subscript is 3, control passes to the statement identified by the label, L3, the third label in the list.

The ALGOL 60 general definition of a switch is not allowed. The labels must all be declared previous to the switch declaration itself. Labels in subblocks of an outer block may not be objects of switches declared in the outer block. When the switch designator is undefined (in the example, when $I \leq 0$ or $I \geq 5$), the GO TO statement is equivalent to a dummy statement.

EQUATE

The EQUATE declaration lets the programmer assign values to certain identifiers. The form of the EQUATE declaration is:

```
EQUATE variable ← constant, ..., variable ← constant
```

EXAMPLE:

```
EQUATE N ← 25, PI ← 3.14159, EPSILON ← '-38;
```

EQUATE identifiers may be used anywhere in a program where a constant can be used. Identifiers are interpreted the same as the constant to which they are equated.

EXAMPLES:

```
ARRAY  A  [1:N, 1:N];  
AREA ← PI * R * R;  
IF X < EPSILON THEN GO TO EXIT;
```

PROGRAM STRUCTURE

The EQUATE identifier PI is recognized by the compiler without having to be declared; its value is equal to 3.14159.

BLOCKS

Declarations define certain properties of the quantities used in the program, and associate them with identifiers. An ALGOL program is segmented into units called blocks, and a declaration of an identifier is valid for the block in which it appears. The particular identifier may be used outside this block for other purposes.

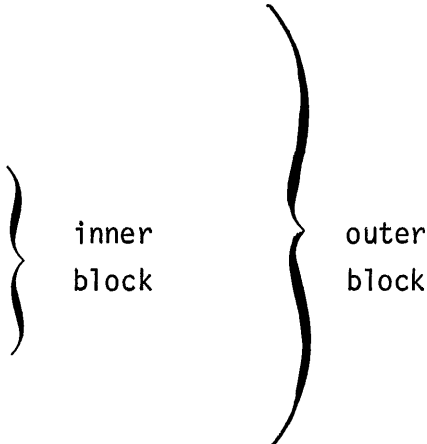
A block begins with the symbol BEGIN and ends with the symbol END. Following the BEGIN are a number of declarations, then a number of statements, all separated by semicolons:

```
BEGIN D; D; ...; D; S; S; ...; S END
```

Because a block is a type of statement, each statement may be a block and must be followed by a semicolon, when appropriate.

EXAMPLE:

```
BEGIN INTEGER I,J,K;
      REAL X,Y;
      S;S;
      BEGIN INTEGER I;
        REAL J,L;
        S;S
      END;
      S; S;S
END
```



The identifiers K, X, and Y are declared in the outer block and are valid throughout. Since the inner block is also part of the outer block, K, X, and Y are also valid there. The identifier L is declared for the inner

PROGRAM STRUCTURE

block only and is meaningless outside it. The identifiers I and J are declared in both outer and inner blocks and may have different definitions in the two blocks. Statements in the inner block which reference I and J refer to the variables declared in the inner block. Statements in the outer block (but not also in the inner block) refer to the variables declared in the outer block.

ARITHMETIC EXPRESSIONS

An arithmetic expression is a rule for computing a numerical value. In simple arithmetic expressions, this value is obtained by executing the indicated arithmetic operations on the values of the quantities involved. The value of a constant is its value; the value of a variable is the last value assigned to it; the value of a function designator is the value arising from the computational rules defining that function procedure.

NOTE: Integer overflow resulting from arithmetic operations is not detected at execution time.

Operators and Types

Arithmetic expressions must be type REAL or type INTEGER. The meaning of these operators is as follows:

The operators +, -, and * have the standard mathematical meaning (addition, subtraction, and multiplication). The type of an expression is INTEGER if both of the operands are integer, otherwise type is REAL.

The operator / also has the conventional mathematical meaning, but the resultant expression type is always REAL.

The operator \ (in ALGOL 60, ÷) is defined only for integer operands and results in remainderless division. The value is defined as follows:

$$A \setminus B = \text{SIGN } (A/B) * \text{ENTIER } (\text{ABS } (A/B))$$

PROGRAM STRUCTURE

EXAMPLES:

$$2 \setminus 1 = 2$$

$$3 \setminus 2 = 1$$

$$(-3) \setminus (-2) = 1$$

$$(-24) \setminus 5 = -4$$

$$7 \setminus 8 = 0$$

The value always is of type INTEGER.

The operator MOD may be used to compute the remainder of an integer division.

It is defined as follows:

$$A \text{ MOD } B = A - B * (A \setminus B)$$

EXAMPLES:

$$2 \text{ MOD } 1 = 0$$

$$3 \text{ MOD } 2 = 1$$

$$(-3) \text{ MOD } (-2) = -1$$

$$(-24) \text{ MOD } 5 = -4$$

$$7 \text{ MOD } 8 = 7$$

The operator $\uparrow$ denotes exponentiation, and is defined as follows:

$a \uparrow i$ (i of type INTEGER):

if $i > 0$, $a * a * \dots * a$ (i times), of the same type as a .

if $i = 0$, if $a \neq 0$, 1, of the same type as a .

if $a = 0$, undefined

if $i < 0$, if $a \neq 0$, $1/(a \uparrow (-i))$, of type REAL.

if $a = 0$, or a of type INTEGER, undefined.

$a \uparrow r$ (r of type REAL):

if $a > 0$, $e^{r \ln(a)}$, of type REAL

if $a = 0$, if $r > 0$, 0.0, of type REAL

if $r \leq 0$, undefined

if $a < 0$, undefined

PROGRAM STRUCTURE

The order of operations is determined by the parenthetical structure of the expression. In the absence of parentheses, the operators are executed in the following order:

↑ (first)
*/ \ MOD (second)
+- (third)

Operators in the same group are evaluated from left to right.

EXAMPLES:

$$A + B * C * D / E \uparrow F \uparrow G \quad \text{value} = A + \frac{B \times C \times D}{(E^F)^G}$$
$$(A + B) * C + D / E \uparrow (F \uparrow G) \quad \text{value} = (A + B) \times C + \frac{D}{E^{(F^G)}}$$

The operators AND, OR, and NOT act upon the entire 16 bits of a word even though the type BOOLEAN uses only one bit.

Conditional Expressions

Another type of arithmetic expression has the following form:

IF <condition> THEN <expression₁> ELSE <expression₂>

This expression is evaluated as follows: if the condition is true, the value of the expression is <expression₁>. If the condition is false, the value is <expression₂>.

NOTE: Integer overflow resulting from arithmetic operations is not detected at execution time. Care must be taken when relational operators are used with integer operands.

Each of the subexpressions may have the IF form.

EXAMPLE:

IF A<B THEN IF A<C THEN A ELSE C ELSE
IF B<C THEN B ELSE C

The value of this expression is minimum {A,B,C}.

PROGRAM STRUCTURE

The form may be enclosed in parentheses and combined with other expressions.

EXAMPLES:

```
IF A<B THEN A + C ELSE B + C
(IF A<B THEN A ELSE B) + C
```

Note that the value of the two expressions is the same.

VARIABLES

There are three types of operands in arithmetic expressions -- constants, function designators and variables. There are subscripted variables and simple variables, each of which may be of types INTEGER or REAL.

Simple Variables

A simple variable is a single value. This value can be used in arithmetic expressions for forming other values, and is changed by assignment statements. The type of the value of a particular variable is defined in the declaration for the variable itself.

Subscripted Variables

Subscripted variables designate values which are components of multidimensional arrays. A subscripted variable has the form:

<identifier> [<subscript expression> , ... , <subscript expression>]

The number of subscript expressions must be identical to the number of dimensions specified in the array declaration. Each subscript must have an integer value which lies within the boundaries specified for the array in the array declaration.

PROGRAM STRUCTURE

EXAMPLES:

R [3, 5, 7]

ABC [I + 2, I * J + K, ENTIER (X), M [I, J, K]]

ASSIGNMENT STATEMENTS

Assignment statements are used to assign the value of an expression to one or more variables or to procedure identifiers. The general form is:

$$\langle \text{identifier}_1 \rangle \leftarrow \langle \text{identifier}_2 \rangle \leftarrow \dots \leftarrow \langle \text{expression} \rangle$$

Assignment to a procedure identifier may be made only within the body of a procedure defining the value of a function designator.

EXAMPLES:

A $\leftarrow$ B

A $\leftarrow$ B $\leftarrow$ C + D - E

A $\leftarrow$ B [I, J] $\leftarrow$ C [3, N] - C [K, N + I]

The variables which precede the assignment symbol ($\leftarrow$) are called left part variables. The arithmetic expression following the last assignment symbol is computed and is then assigned to each of the left part variables in turn, from right to left.

Types

In contrast to ALGOL 60, the left part variables need not all be of the same type. If a value of one type is to be assigned to a variable of a different type, the appropriate transfer function is applied.

PROGRAM STRUCTURE

In the case of a REAL value being assigned to an INTEGER variable, the transfer function has the value

ENTIER (<expression> + 0.5)

When the types of the left part variables differ, transfer functions are applied from right to left. For example, let X and Y be of type REAL, and I be of type INTEGER. Then the statement:

$X \leftarrow I \leftarrow Y \leftarrow .3$

will cause the following assignments to take place:

$Y \leftarrow .3$
 $I \leftarrow 0$
 $X \leftarrow 0.0$

Note that HP ALGOL uses the symbol "<" instead of the ALGOL 60 symbol " : = ". However, the symbol " : = " will be recognized as being equivalent.

GO TO STATEMENTS

A GO TO statement interrupts the normal sequence of operations, which is the lexical order of statements. The general form is:

GO TO <designational expression>

GO TO defines its successor explicitly by the value of a designational expression. The next statement to be executed will be the one having this value as a label.

EXAMPLES:

GO TO L
GO TO S [I]

PROGRAM STRUCTURE

Designational Expressions

A designational expression is an expression whose value is a label of a statement. There are two types of designational expressions, labels and switch designators. The ALGOL 60 IF type of designational expression is not allowed.

Labels

A label is an identifier used to name a statement as one to which control may be passed by a GO TO statement. More than one label may be used for a given statement. The general form is:

`<label> : <label> : ... : <label> : S`

Unsigned integers may not be used as labels.

EXAMPLES:

`L: A ← X + 1`
`L1: L2: GO TO S`

Every label must appear in a LABEL declaration in the innermost block in which it appears.

A GO TO statement may not refer to a label in a sub-block of the block in which the GO TO statement appears. As a result, a block may be entered only at its head.

CONDITIONAL STATEMENTS

Conditional statements cause certain statements to be executed or skipped depending upon certain logical conditions.

PROGRAM STRUCTURE

There are two general forms:

IF <condition> THEN S_1 ELSE S_2

IF <condition> THEN S

In the first form, if the condition is true, then statement S_1 is executed and S_2 skipped. Otherwise, S_2 is executed and S_1 skipped. The IF statement is considered a single statement and should include no semicolons.

In the second form, if the condition is true, S is executed. If it is false, S is skipped.

EXAMPLES:

IF $A < B$ THEN $A \leftarrow B$ ELSE $B \leftarrow A$

IF $A < B$ THEN $A \leftarrow A + 1$

IF $A < B$ THEN GO TO L ELSE IF $B < C$
THEN GO TO M ELSE $A \leftarrow B \leftarrow C \leftarrow 0$

Each of the statements (following either THEN or ELSE) may itself be a conditional statement. If there are fewer ELSEs than IFs, the definition of the conditional statement is ambiguous. This is resolved in HP ALGOL by associating each ELSE with the closest preceding unmatched IF.

For example, the statement

IF $A < B$ THEN IF $B < C$ THEN S_1 ELSE S_2

is interpreted as

IF $A < B$ THEN
{ IF $B < C$ THEN S_1 ELSE S_2 }

rather than as

IF $A < B$ THEN { IF $B < C$ THEN S_1 } ELSE S_2

PROGRAM STRUCTURE

BOOLEAN EXPRESSION

A Boolean expression is one which has a logical value of TRUE or FALSE. It consists of one or more operands associated with one or more logical operators.

EXAMPLES:

X = -2
Y > V OR Z < Q
A + B >= -5 AND Z - D > Q + 2
A > B AND A > C AND A > D
NOT (A > B OR A > C)

The values of these expressions can be determined from the following table:

<u>B₁</u>	<u>B₂</u>	<u>B₁ OR B₂</u>	<u>B₁ AND B₂</u>	<u>NOT B₁</u>
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

The complete hierarchy of operators is as follows:

↑
*/\MOD
+-
< <= = >= > #
NOT
AND
OR

Note that the ALGOL 60 operators $\supset$ (implies) and $\equiv$ (equivalent) have been eliminated due to lack of usage.

PROGRAM STRUCTURE

Boolean Variables

Just as arithmetic values can be assigned to REAL or INTEGER variables, Boolean values can be assigned to variables of type BOOLEAN. For example, if B_1 and B_2 are of type BOOLEAN, and A_1 , A_2 , and A_3 are of type REAL, the following statements are all legitimate:

```
B1 ← A1 < A2
IF B1 THEN GO TO L
IF B1 OR A1 = A2 AND A2 = A3 THEN B2 ← TRUE
B1 ← B2 ← NOT B1
B1 ← IF B2 THEN A1 = A2 ELSE A1 = A2 AND B1
```

Boolean variables can also be subscripted, just like arithmetic variables.

EXAMPLE:

```
B1 [ I, J, K ] ← NOT B1 [ I - 1, J - 1, K - 1 ]
A1 ← IF B1 [ I ] THEN A1 ↑ 2 ELSE IF B1 [ I + 1 ]
      THEN A1 ↑ 3 ELSE 0
```

To the HP ALGOL compiler, there is no difference between Boolean and integer variables. As a result, Boolean operators and arithmetic operators may be mixed within an expression.

Integer values are considered to be TRUE when they are negative and FALSE when they are positive or zero: thus, only the sign bit (bit 15) is significant with respect to truth or falsity. The operators AND, OR, and NOT act upon all 16 bits, using the machine instructions AND, IOR, and CMA .

The constants TRUE and FALSE are in all ways equivalent to -1 and 0. The two expressions following are different:

```
IF <Boolean expression> THEN...
IF< Boolean expression> = TRUE THEN...
```

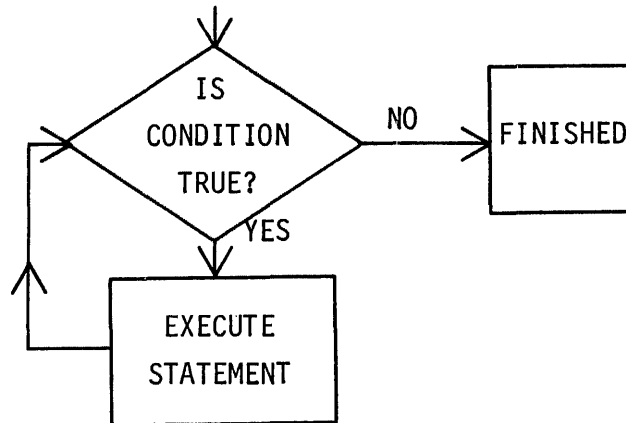
PROGRAM STRUCTURE

WHILE STATEMENTS

The WHILE statement causes repeated execution of a statement as long as a condition exists. The general form is:

WHILE <condition> DO <statement>

When the condition is satisfied, control passes to the next statement in sequence. The processing flow of a WHILE statement is as follows:



EXAMPLE:

```
WHILE A < B DO
  BEGIN A ← A + 1; B ← B - 1 END
```

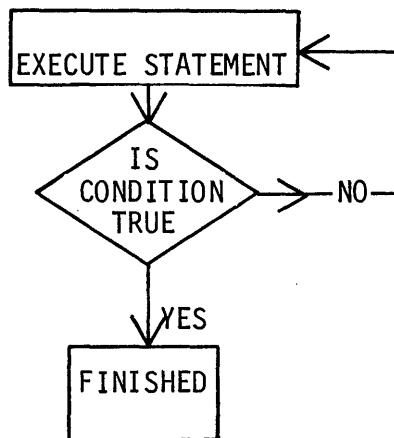
Note that the condition is tested before execution, so that if the condition is not TRUE, the statement will not be executed--not even the first time the condition is examined.

DO STATEMENT

The DO statement causes repeated execution of a statement UNTIL a condition becomes TRUE.

DO <statement> UNTIL <Boolean expression>

PROGRAM STRUCTURE



The statement is executed first, then the Boolean expression is tested. If the expression is FALSE, the statement is reexecuted. If the statement is TRUE, flow continues. *Note that the DO statement will be executed at least once whether the expression is TRUE or FALSE.*

COMPOUND STATEMENTS

It is often necessary to treat a number of statements as a single logical unit. A typical case is when the execution of the set of statements is subject to a condition. This is easily accomplished in ALGOL by use of the compound statement. A compound statement has the form:

BEGIN S; S; ... S END

EXAMPLE:

```
IF A < B THEN
  BEGIN A ← B;
        B ← B + 1
  END ELSE
  BEGIN B ← A; A ← A + 1 END
```

This example is a single conditional statement. First the condition $A < B$ is tested. If it is true, then the statements $A \leftarrow B$ and $B \leftarrow B + 1$ are executed. If it is false, the statements $B \leftarrow A$ and $A \leftarrow A + 1$ are executed.

PROGRAM STRUCTURE

Since any S can also be a compound statement, they can be nested indefinitely.

EXAMPLE:

```
I ← 0;
IF A < 0 THEN
  BEGIN I ← 1;
    IF B < 0 THEN
      BEGIN
        I ← 2;  IF C < 0 THEN I ← 3
      END
    END
  END
```

This example sets I to one of the following values:

```
I ← 0  if A ≥ 0
      1  if A < 0, B ≥ 0
      2  if A < 0, B < 0, C ≥ 0
      3  if A < 0, B < 0, C < 0
```

FOR STATEMENTS

The FOR statement allows repeated execution of a statement while performing a sequence of assignments to a "control" variable within the statement. The general form is:

```
FOR <integer variable> ← <initial value> STEP <increment> UNTIL <final value>
  DO <statement>
```

The variable is assigned the initial value, the statement is executed, the variable incremented, the statement executed again, etc. Execution terminates when the control variable exceeds the final value. The step value and final value are evaluated once upon entering execution of the statement. They are not re-evaluated for subsequent steps. The initial value, the increment, and the final value can be any expressions, including negative ones.

NOTE: Integer overflow resulting from arithmetic operations is not detected at execution time.

PROGRAM STRUCTURE

The following statement is a simple example. It sets the first n elements of a one-dimensional array to zero:

```
FOR I ← 1 STEP 1 UNTIL N DO A [I] ← 0
```

The effect is to repeat the execution of the statement following the symbol DO ($A[I] \leftarrow 0$) with the variable I changing its value each time. It takes on the values

1, 2, 3, ... , N .

This FOR statement is equivalent to:

```
I ← 1 ; A [I] ← 0;  
I ← 2 ; A [I] ← 0;  
...  
I ← N ; A [I] ← 0;
```

FOR statements can be nested. The following statement is the two-dimensional analogue of the previous example:

```
FOR I ← 1 STEP 1 UNTIL N DO  
  FOR J ← 1 STEP 1 UNTIL N DO A[I,J] ← 0
```

This is equivalent to:

```
A[1, 1] ← 0;  
A[1, 2] ← 0;  
...  
A[1, N] 0;  
A[2, 1] 0;  
A[2, 2] 0;  
...  
A[2, N] 0;  
...  
...
```

PROGRAM STRUCTURE

```
A[N, 1] ← 0;  
A[N, 2] ← 0;  
  . . .  
A[N, N] ← 0;
```

Some examples will serve to illustrate the possibilities of the step-until element.

```
I ← 1 STEP 1 UNTIL 10 (specifies the values 1, 2, 3, ..., 10)  
I ← 10 STEP -1 UNTIL 1 (specifies the values 10, 9, 8, ..., 2, 1)  
I ← 1 STEP 5 UNTIL 10 specifies the values 1, 6)  
I ← 1 STEP 1 UNTIL N  
    (If N > 0; the values specified are 1, 2, 3, ..., N.  
     If N ≤ 0, the FOR list has no values at all.)
```

If the stepping value is 1, the symbols STEP 1 UNTIL may be replaced by the symbol TO. For example,

```
FOR I ← 1 STEP 1 UNTIL N  
    and  
FOR I ← 1 TO N
```

are equivalent.

CASE STATEMENT

The CASE statement causes the execution of one of a number of statements following the CASE statement:

```
CASE <expression>  
BEGIN <statement 1> ;  
      <statement 2> ;  
      ⋮  
      <statement n>  
END
```

PROGRAM STRUCTURE

The expression will be evaluated. If its value is between 1 and n, the statement corresponding to that value will be executed. If the value is not between 1 and n, the entire CASE statement is bypassed.

PAUSE STATEMENT

The PAUSE statement permits the operator to perform some action such as turning on (or off) some device.

For SIO, a PAUSE statement causes the word "PAUSE" to be printed on the teleprinter and the computer to be brought to a halt. Program execution resumes when the RUN button is pressed.

For DOS, a PAUSE directive causes the current job to be suspended until the operator enters a GO directive. A message for the operator may be included in the directive.

For RTE, a PAUSE statement causes a specified job to be suspended until the operator requests GO.

SECTION II

PROCEDURES

Statements or blocks describing common computational processes may occur several times in the same program, perhaps with different names used to designate some of the quantities involved. Such a process, called a procedure, is often designated and called into execution by a special declaration.

PROCEDURE DECLARATIONS

The PROCEDURE declaration defines a process, as described above. A procedure is subject to the same rules of validity as any declared variable. (See Section I.)

A procedure contains a procedure body which consists of a single statement, most often a block. Associated with the procedure body is the procedure heading which specifies the parameters to the procedure.

EXAMPLE:

Procedure Heading	[	PROCEDURE TRANSPOSE (A,N); VALUE N; INTEGER N; ARRAY A;
Procedure Body	[	BEGIN REAL Z; INTEGER I, J; FOR I ← 1 STEP 1 UNTIL N DO FOR J ← I + 1 STEP 1 UNTIL N DO BEGIN Z ← A[I,J]; A[I,J] ← A[J,I]; A [J,I] ← Z END END
	]	END

PROCEDURES

This sample procedure transposes an $n \times n$ matrix. The procedure heading consists of the following parts:

The reserved word: PROCEDURE

The procedure identifier: TRANSPOSE

The formal parameter part: (A,N);(optional)

The value part: VALUE N; (optional)

The specification part: INTEGER N; ARRAY A; (not required if there are no formal parameters)

The reserved word PROCEDURE identifies a procedure declaration. The procedure identifier calls the procedure into execution at another place in the program. The formal parameter part gives the names of those identifiers used within the procedure body, but which, when the procedure is called, are replaced by actual parameters whose names may be different. The value part indicates those formal parameters to be called by value. The specification part indicates the types of the formal parameters. In HP ALGOL, all the formal parameters must be specified.

There are no restrictions on the types of formal parameters. Thus, they may be any of the following:

REAL	BOOLEAN ARRAY	LABEL
INTEGER	REAL PROCEDURE	SWITCH
BOOLEAN	INTEGER PROCEDURE	FORMAT
ARRAY	BOOLEAN PROCEDURE	INPUT
REAL ARRAY	PROCEDURE	OUTPUT
INTEGER ARRAY		

The procedure body may contain references to any formal parameters, to any local variables (those declared in the procedure body itself, if it is a block), and to any variables declared outside the procedure declaration.

PROCEDURES

PROCEDURE STATEMENTS

A procedure is called by a PROCEDURE statement.

EXAMPLE:

TRANSPOSE (X,25)

This statement calls the procedure which was declared with the name TRANSPOSE. The parameters in parentheses are called actual parameters, and they must correspond in number and type to the formal parameters specified in the PROCEDURE declaration.

Note the correspondence in the example:

<u>FORMAL PARAMETER</u>	<u>ACTUAL PARAMETER</u>	<u>TYPE</u>
A	X	array
N	25	integer

When the statement TRANSPOSE (X,25) is executed, the elements in the array X are transposed, as follows: if the value of a given element of X, say $X[I,J]$, was X_{ij} before the statement is executed, then after the statement is executed, the value is $X[I,J] = X_{ji}$. This is accomplished by substituting the array X for A wherever A appears in the procedure body. This substitution is the meaning of a parameter which is called by name, not called by value.

CALL BY NAME/VALUE

Parameters called by value are those whose names appear in the value part of the procedure heading. The main distinction is that formal parameters called by value are computed when the procedure is called and are treated as local variables. Assignments to these parameters have no effect on the value of the actual parameter.

PROCEDURES

In parameters called by name, the actual parameter is substituted for the formal parameter wherever the latter appears in the procedure body. As a result, any assignments to the formal parameter do affect the value of the actual parameter. When a type parameter is called by name, the actual parameter must be either a simple variable or a subscripted variable.

EXAMPLE:

```
PROCEDURE P1(X); REAL X; X ← X + 1;
PROCEDURE P2(X); VALUE X; REAL X; X ← X + 1;
A ← 0;
P1 (A);
P2 (A);
```

After the procedure P1 is called, the value of A becomes 1.0. It is changed because P1 specifies that the parameter be called by name. If P2 is called next, the value of A is still 1.0 since P2 specifies that its parameter is called by value.

EXAMPLE:

```
INTEGER N, Z;
:
PROCEDURE SAMPLE (M, Y);
VALUE M; INTEGER M, Y;
BEGIN
    INTEGER A;
    A ← M;
    Y ← A + 1;
    M ← M + 1
END
N ← 3; Z ← 1;
:
SAMPLE (N, Z)      actual parameters
                   Result: Z ← N + 1
```

The value of N is unchanged because M was called by value.

PROCEDURES

Only parameter types REAL, INTEGER, and BOOLEAN may be called by value.

(Arrays are not called by value because storage is not allocated dynamically.)

If an actual parameter is a procedure, then all the parameters of that procedure are called by value.

No procedure may be entered recursively, either implicitly or explicitly.

If an actual parameter is itself a procedure, then all of its parameters must be called by value.

FUNCTION PROCEDURES

A function procedure is a procedure that results in a single value. This value must be assigned somewhere in the procedure body by an assignment statement with the procedure identifier on the left-hand side.

EXAMPLE:

```
REAL PROCEDURE TRACE (A,N);  
  VALUE N; INTEGER N; ARRAY A;  
BEGIN REAL S; INTEGER I;  
  S ← 0; FOR I ← 1 STEP 1 UNTIL N DO S ← A[I,I] + S;  
  TRACE ← S  
END
```

To specify a function procedure, the word PROCEDURE must be preceded by the type (REAL, INTEGER, or BOOLEAN). A function procedure is called by a PROCEDURE statement or an expression. When a function procedure call appears in an expression, its value is treated as that of a variable of the same type.

PROCEDURES

CODE PROCEDURES

An ALGOL procedure may be compiled separately from a main program or be written in Assembly Language or FORTRAN. These procedures may be referenced from an ALGOL program by replacing the procedure body with the word CODE. The specifications must all be given.

EXAMPLES:

```
PROCEDURE INVER (A, X, N);  
VALUE N; INTEGER N; REAL A, X; CODE;  
REAL PROCEDURE INTEGRAL (A, B, F);  
VALUE A, B; REAL A, B: REAL PROCEDURE F; CODE;
```

A CODE procedure identifier may be a maximum of five characters. If there are more than five characters, only the first five are significant.

INTRINSIC PROCEDURES

Certain identifiers are reserved for the standard functions of analysis, and are referenced as procedures:

<u>NAME</u>	<u>MEANING</u>	<u>TYPE</u>
ABS (E)	absolute value of E	Same as that of E
SIGN (E)	=1 if E>0, 0 if E = 0, -1 if E < 0	Integer
SQRT (E)	$\sqrt{E}$	Real
SIN (E)	sin E	Real
COS (E)	cos E	Real
ARCTAN (E)	$\tan^{-1} E$	Real
TANH (E)	tanh E	Real
LN (E)	ln E	Real
EXP (E)	e^E	Real
ENTIER (E)	the largest integer $\leq E$	Integer
ROTATE (I)	rotate I 8 bits (swap half-words)	Integer
KEYS	16-bit value of switch register	Integer
TAN (E)	tan E	Real

SECTION III

INPUT/OUTPUT

HP ALGOL input/output operations involve the following:

- Input Lists
- Output Lists
- Formats
- READ Statements
- WRITE Statements
- Magnetic Tape Statements

LIST DECLARATIONS

I/O lists specify that variables be read or written by a READ or WRITE statement. The general form of an input or output list declaration is:

```
INPUT    <list identifier> ( <list element>, ..., <list element>);
OUTPUT   <list identifier> ( <list element>, ..., <list element>),
        ...      ;
```

Each list identifier refers to a list of elements; each element may be one of the following:

- a. simple variable,
- b. subscripted variable,
- c. list,
- d. expression (only in an OUTPUT list), or
- e. FOR element.

A FOR element is like a FOR statement except that it is followed by either a single list element or a group of list elements. A group of list elements is enclosed in brackets. An input (output) list may call on other input (output) lists. All the elements appearing in a list must be previously declared.

INPUT/OUTPUT

EXAMPLE:

```
INPUT  IN1  (A, B, C, D [I,J], FOR I ← 1 TO L
        DO [M[I], N[I] ], X),
        IN2 (X, Y, Z, IN1);

OUTPUT TRIG(FOR I ← 1 TO L
        DO [I, A[I], SIN(A[I]), COS(A[I])]);
```

FORMAT DECLARATIONS

A FORMAT specification describes the physical arrangement of data. FORMAT specifications must be defined in the declaration part. The general form is:

```
FORMAT <format identifier> (spec1, ..., r(specn, ...), specn, ...),
    <format identifier> ...;
```

More than one format may appear in a FORMAT declaration.

Formats may be declared directly in a READ or WRITE statement, eliminating the need for separate FORMAT statements. See the example of In-Line FORMATS under "READ AND WRITE STATEMENTS" in this section.

Carriage control is required for line printer output. See "CARRIAGE CONTROL" at the end of this section.

FORMAT SPECIFICATIONS

The data elements in the input/output lists may be converted from external to internal and from internal to external representation according to format conversion specifications. If the variable type in the input/output list does not correspond to the type specified in the FORMAT declaration, the compiler converts one type to the other. FORMAT declarations may also contain editing codes.

INPUT/OUTPUT

Conversion Specifications

rEw.d	Real number with exponent
rFw.d	Real number without exponent
rIw	Decimal integer
rAw	Alphanumeric
r@w } rKw }	Octal integer

Editing Specifications

nX	Blank field descriptor
nHh ₁ h ₂ ...h _n } r"h ₁ h ₂ ...h _n " }	Heading and labeling descriptors
r/	Begin new record

Both w and n are positive integer constants representing the width of the field in the external character string; n may be omitted if the width is one. The symbol d is a non-negative integer constant representing the number of digits in the fractional part of the string. The repeat count, r, is an optional positive integer constant indicating the number of times to repeat the succeeding basic field descriptor. Each h is one character.

Ew.d Output

The E specification converts numbers in storage to character form for output. The field occupies W positions in the output record; the number appears in floating point form right-justified in the field as:

$$\Lambda.x_1 \dots x_d E_{\pm ee}$$

The caret symbol, Λ , indicates a space.

INPUT/OUTPUT

$X_1 \dots X_d$ are the most significant digits of the value of the data to be output, while ee are the digits in the exponent. Field W must be wide enough to contain significant digits, signs, decimal point, E , and exponent. Generally, W should be greater than or equal to $d + 7$.

If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs (\$) are inserted in the entire field. If the field is longer than the output value, the quantity is right-justified with spaces to the left.

EXAMPLES:

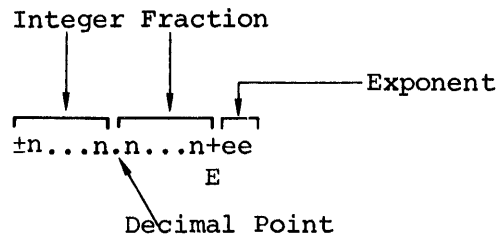
FORMAT F5(E10.3);	
:	A contains +12.34 or -12.34
WRITE(4, F5, A);	Result is $\Lambda\Lambda. 123E+02$ or $\Lambda-.123E+02$
FORMAT F5(E12.3);	
:	A contains +12.34 or -12.34
WRITE(4,F5,A);	Result is $\Lambda\Lambda\Lambda\Lambda. 123E+02$ or $\Lambda\Lambda\Lambda-. 123E+02$
FORMAT F5(E7.3);	
:	A contains +12.34 or -12.34
WRITE(4,F5,A);	Result is $.12E+02$ or $-. 1E+02$
FORMAT F5(E5.1);	
:	A contains +12.34
WRITE(4,F5,A);	Result is \$\$\$\$

Ew.d Input

The E specification converts the number in the input field (specified by W) to a real number and stores it in the appropriate storage locations.

INPUT/OUTPUT

The input field may consist of integer, fraction, and exponent subfields:



The integer subfield begins with a + or - sign or a digit and may contain a string of digits terminated by a decimal point, an E, +, -, or the end of the input field.

The fraction subfield begins with a decimal point and may contain a string of digits terminated by an E, +, -, or the end of the input field.

The exponent field may begin with a sign or an E and contains a string of digits. When it begins with E, the + is optional between E and the string. The value of the string of digits should not exceed 38. The number may appear in any position within the field; spaces in the field are ignored.

EXAMPLES:

+1.2345E2
123.456+9
-0.1234-6
.12345E-3
1234
+12345
+1234E6

When no decimal point is present in the input quantity, d acts as a negative power of ten scaling factor. The internal representation of the input quantity will be:

(Integer Subfield) $\times 10^{-d}$ $\times 10$ (Exponent Subfield)

INPUT/OUTPUT

EXAMPLE:

```
FORMAT F(E12.8);      Input quantity =  $\Lambda\Lambda\Lambda 1234+5\Lambda\Lambda$   
Conversion performed:  $1234 \times 10^{-8} \times 10^5$   
Result:                1.234
```

If a d value in the specification conflicts with a decimal point appearing in an input field, the actual decimal point takes precedence.

EXAMPLE:

```
FORMAT F(E12.8);      Input quantity =  $\Lambda\Lambda\Lambda\Lambda\Lambda 1.234+5$   
Quantity stored:       $1.234 \times 10^5$ 
```

The field width specified by w should always be the same as the width of the input field. When it is not, incorrect data may be read, converted and stored. The value of w should include positions for signs, the decimal point, the letter E, as well as the digits of the subfields:

EXAMPLE:

```
FORMAT F10(E7.2,E5.3,E9.2);  
:  
READ(5,F10,A,B,C);  
Assuming input data in contiguous fields:  
-12.3E1+1234123.46E-3  
|←E7 →|←5 →|← 9 →|
```

The fields read would be:

```
-12.3E1  
+1234  
123.46E-3
```

INPUT/OUTPUT

and converted as:

-123.
1.234
.12346

However, if specifications were:

FORMAT F10(E7.2,E4.3,E7.2);

The fields read would be:

-12.3E1
+123
4123.46

and converted as:

-123
.123
4123.46

The effects of possible FORMAT specification errors such as the above may not be detected by the system.

EXAMPLE:

<u>FORMAT Specification</u>	<u>Input Field</u>	<u>Converted Value</u>
E9.2	+1.2345E2	123.45
E9.4	-0.1234-6	-.0000001234
E4.2	1234	12.34

INPUT/OUTPUT

Fw.d Output

The F specification converts real numbers in storage to character form for output. The field occupies w positions and will appear as a decimal number, right-justified in the field:

ΛX...X.X...X

The x's are the most significant digits. The number of decimal places to the right of the decimal point is specified by d. If d is zero, no digits appear to the right of the decimal point.

The field must be wide enough to contain the significant digits, sign and decimal point. If the number is positive, the + sign is suppressed. If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs (\$) are inserted in the entire field. If the field is longer than the output value, the number is right-justified with spaces occupying the excess positions on the left.

EXAMPLES:

{	FORMAT F5(F10.3);	
	:	A contains + 12.34 or -12.34
{	WRITE(4,F5,A);	Result: ΛΛΛΛ12.340 or ΛΛΛΛ ⁻ 12.340
	FORMAT F5(F12.3);	
{	:	A contains +12.34 or -12.34
	WRITE(4,F5,A);	Result: ΛΛΛΛΛΛ12.340 or ΛΛΛΛΛΛ ⁻ 12.340
{	FORMAT F5(F4.3);	
	:	A contains +12.34
{		Result: 12.3
	WRITE (4,F5,A);	

INPUT/OUTPUT

{	FORMAT F5(F4.3);	
	:	A contains +12345.12
	WRITE(4,F5,A);	Result: \$\$\$\$

Fw.d Input

The F specification input is identical to the E specification input. Although the fields are generally assumed to contain only a sign, integer, decimal point, and fraction, they may also contain an exponent subfield. All restrictions for Ew.d input apply.

Iw

The Iw specification converts internal values to output character strings or input character strings to internal numbers. The output external field occupies w record positions and appears right-justified as:

$$\Delta x_1 \dots x_d$$

The x's represent the decimal digits (maximum of 5) of the integer. When the integer is positive on output, the sign is suppressed. If an output field is too short, dollar signs (\$) will be placed in the output record.

The Iw specification, when used for input, is identical to an Fw.0 specification.

INPUT/OUTPUT

EXAMPLES:

<pre> FORMAT F10(I5,I5,I4,I6); . . WRITE(6,F10,I,J,K,L); Result: -123412345\$\$\$\$ 12345 ← 5→ ←5→ ←4→ ←6→ FORMAT F10(I5,I5,I4,I1); . . READ(5,F10,I,J,K,L); I contains -0123 J contains 12003 K contains 0102 L contains 3 </pre>	<pre> I contains -1234 J contains +12345 K contains +12345 L contains +12345 Input contains: -11231211311123 ←5 → ← 5→ ←4→ ←1 </pre>
--	---

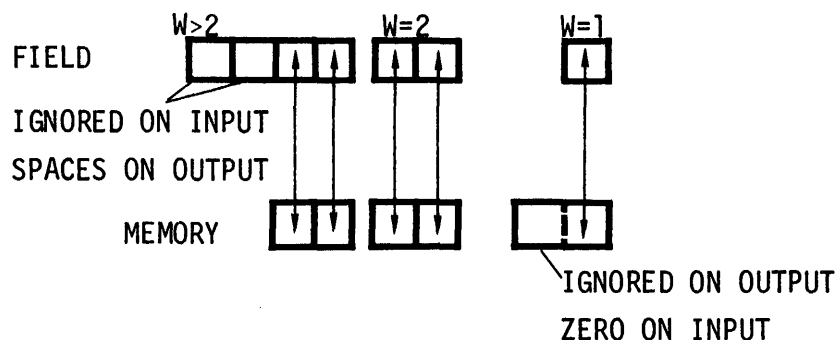
Aw

The Aw specification causes alphanumeric data on an external medium to be translated to or from ASCII form in memory. The associated list element must be of type INTEGER.

On input, if the field indicated by w is greater than 2, the first w-2 characters are ignored; only the last two characters are read. When w equals 2, the two characters are read. If w equals 1, one character is read and stored in the left half of a computer word; blank is entered in the right half.

On output, if the field is greater than 2, two characters are written right-justified in the field; the leading positions are filled with spaces. If w equals 2, the two characters are written. If w equals 1, the character in the left half of the computer word is written.

INPUT/OUTPUT



EXAMPLE:

```

Input data:  AZZ213-ABCXABC137-ZZ9  (CR) (LF)
INTEGER ARRAY ID[1:5]; INTEGER I2, I1, I;
FORMAT F10(A10,A1,5A2);

:
:
:
:
READ(5,F10,I2,I1,ID);FOR I=1 TO 5 DO ID [I]);
Result:  I2  BC
         I1  0X
         ID  AB
           C1
           37
          -Z
          Z9
    
```

@w and Kw

Octal integer values are converted under either the @ or the K specification. The field is w octal digits in length; the corresponding list element must be INTEGER.

On input, if w is greater than or equal to 6, up to six octal digits are stored; non-octal digits appearing within the field are ignored. If the value of the octal digits within the field is greater than 177777, the

INPUT/OUTPUT

results are unpredictable. If w is less than 6 or if fewer than six octal digits are encountered in the field, the number is right-justified in the computer word and filled with leading zeros.

On output, if the field is greater than 6, six octal digits are written right-justified in the field; the leading positions are filled with spaces. If w equals 6, the six octal digits are written. If w is less than 6, the w least significant octal digits are written.

EXAMPLE:

```
Input data: 123456-1234562342342342,396E-05 (CR) (LF)
INTEGER ARRAY ID[1:2], IE[1,2]; INTEGER IB, IC;
FORMAT F10(06, 07, 205, 204);

:
READ(5,F10,IB,IC,ID[1]; ID[2], IE[1], IE[2]);
Result:  IB 123456
          IC 123456
          ID 023423
           042342
          IE 0000036
           0000005
```

nX

The X specification is used to include n blanks in an output record or to skip n characters on input to permit spacing of input/output quantities. In the specifications list, the comma following $\wedge X$ is optional. $\wedge X$ is interpreted as $1X$; $0X$ is not permitted.

INPUT/OUTPUT

EXAMPLE:

```

FORMAT F10(E8. 3, 5X,F6.2,5X,I4);  A contains + 123.4
                                   B contains -12.34
                                   I contains -123
                                   :
WRITE(6,F10,A,B,I);
Result:  .1234E3      -12.34      -123
Input:
WEIGHT  10  PRICE  $1.98  TOTAL  $19.80
FORMAT F10(8X,I2,10X,F4.2,10X,F5.2);
                                   :
READ(5,F10,I,A,B);
Result:  I contains 10
          A contains 1.98
          B contains 19.80

```

nHh₁h₂...h_n

The H specification transfers any combination of eight-bit ASCII characters, including blanks; n is an unsigned integer specifying the number of characters to the right of the H that are to be transmitted. The comma following the H specification is optional. H is interpreted as 1H; 0H is not permitted.

On output, the ASCII data in the FORMAT statement is written in the form of comments, titles, and headings. See "CARRIAGE CONTROL" at the end of this section for line printer driver requirements.

INPUT/OUTPUT

EXAMPLE:

```
FORMAT F10(20H THIS IS AN EXAMPLE );
```

```
:
```

```
WRITE(6,F10);
```

Result: THIS IS AN EXAMPLE

```
FORMAT F10(8HWEIGHT ,I2,10H PRICE $ ,F4.2,11H TOTAL $ ,F5.2);
```

```
:
```

```
WRITE(6,F10,I,A,B);
```

I contains 10

A contains 1.98

B contains 19.80

Result: WEIGHT 10 PRICE \$ 1.98 TOTAL \$ 19.80

On input, the data is transmitted from the unit to the FORMAT statement.

A subsequent output statement transfers the new data to the output record.

EXAMPLES:

```
FORMAT F10(34H );
```

```
:
```

```
READ(5,F10);
```

```
WRITE(6,F10);
```

Input: THIS INPUT ALLOWS VARIABLE HEADERS

Result: THIS INPUT ALLOWS VARIABLE HEADERS

INPUT/OUTPUT

$r"h_1h_2\dots h_n"$

This specification also transfers any combination of ASCII characters (except the quotation marks). The number of characters transmitted is the number of positions between the quotation marks; field length is not specified. If r , an optional repeat count, is present, the character string within the quotation marks is repeated that number of times. Commas preceding the initial quotation mark and following the closing quotation are optional. See "CARRIAGE CONTROL" at the end of this section for line printer driver requirements.

EXAMPLES:

```
FORMAT F10("THIS IS ALSO AN EXAMPLE");
```

.

```
WRITE(6,F10);
```

Result: THIS IS ALSO AN EXAMPLE

```
FORMAT F10(3"ABC");
```

.

```
WRITE(6,F10);
```

Result: ABCABCABC

On input, the number of characters within the quotation marks is skipped on the input field.

- NOTE: 1) If the closing quotation mark is omitted in coding a *FORMAT* statement of this type, results are unpredictable.
- 2) The single quote (apostrophe) is illegal for this usage. It may compile correctly but will generate run-time errors.

NEW RECORD

A slash terminates the current record and signals the beginning of a new record of formatted data. A slash may occur anywhere in the specifications list and need not be separated from the other list elements by commas.

INPUT/OUTPUT

Several records are skipped by indicating consecutive slashes or by preceding the slash with a repetition factor; $r-1$ records are skipped for $r/$. On output, the slash is used to skip lines, cards, or tape records; on input, it specifies that control passes to the next record or card.

EXAMPLES:

```
FORMAT F10(22X,6HBUDGET///  
        6HWEIGHT,6X,5HPRICE,  
        9X, 5HTOTAL, 8X);
```

:

```
WRITE(6,F10);
```

or

```
FORMAT F10(22X,6HBUDGET,3/  
        6HWEIGHT,6X,5HPRICE,  
        9X,5HTOTAL,8X);
```

Result:

```
line 1  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ BUDGET  
line 2  
line 3  
line 4 WEIGHT ^^^^^^^ PRICE ^^^^^^^^^^^ TOTAL ^^^^^^^^^
```

REPEAT SPECIFICATIONS

Field descriptors (except nH) are repeated by preceding the descriptor with a repeat count, r . If the input/output list warrants, the conversion is interpreted repetitively up to the specified number of times.

INPUT/OUTPUT

A group of field descriptors, including nH, is repeated by enclosing the group in parentheses and preceding the left parenthesis with a group repeat count. If no group repeat count is specified, a value of one is assumed. Grouped field descriptors may not be nested.

EXAMPLES:

```
FORMAT F10(I5,I5,I5);
```

```
:
```

```
WRITE(4,F10,I,J,K);
```

can be written as

```
FORMAT F10(3I5);
```

```
:
```

```
WRITE(4,F10,I,J,K);
```

```
FORMAT F10(E8.3,5X,F6.2,5X,I4,E8.3,5X,F6.2,5X,I4);
```

```
:
```

```
WRITE(4,F10,A,B,I,C,D,J);
```

can be written as

```
FORMAT F10(2(E8.3,5X,F6.2,5X,I4));
```

```
:
```

```
WRITE(4,F10,A,B,I,C,D,J);
```

A nested repetition specification would be:

```
FORMAT F10(E8.3,5X,5(F6.2,5X,I4));
```

The group F6.2, 5X,I4 would be written five times, and the entire group, once.

INPUT/OUTPUT

FREE FIELD INPUT

Using certain conventions in the input data, an HP ALGOL program can be written without FORMAT statements. Special symbols included with the ASCII input data items direct the formatting:

space	Data item delimiters
/	Record terminator
+ -	Sign of item
.E+-	Floating point number
@	Octal integer
"..."	Comments

All other ASCII non-numeric characters are treated as spaces (and delimiters). Free field input may be used for numeric data only and is indicated by an asterisk in the READ statement rather than a FORMAT identifier.

Data Item Delimiters

Any contiguous string of numeric and special formatting characters occurring between two commas, a comma and a space, or two spaces, is a data item whose value corresponds to a list element. A string of consecutive spaces is equivalent to one space. Two consecutive commas indicate that no data item is supplied for the corresponding list element; the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

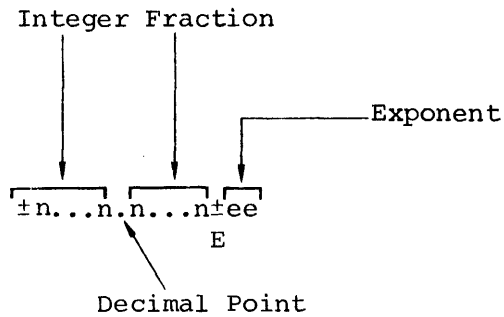
INPUT/OUTPUT

EXAMPLE:

1) READ(5,*,I,J,K,L);	2) READ(5,*,I,J,K,L);
Input data: 1720, 1966/ 1980 1492	Input data: 1266,,1794,2000
Result: I contains 1720 J contains 1966 K contains 1980 L contains 1492	Result: I contains 1266 J contains 1966 K contains 1794 L contains 2000

Floating Point Input

The symbols for a floating point data item are the same as those used to represent floating point data for FORMAT statement directed input.



If the decimal point is not present, it is assumed to follow the last digit.

EXAMPLES:

```
READ(5,*,A,B,C,D,E);  
Input Data: 3.14, 314E-2, 3140-3, .0314+2, .314E1  
All are equivalent to 3.14
```

INPUT/OUTPUT

Octal Input

An octal input item has the following format:

$$@ x_1 \dots x_d$$

The symbol @ defines an octal integer. The x's are octal digits each in the range of 0 through 7. List elements corresponding to the octal data items are INTEGER.

Record Terminator

A slash within a record causes the next record to be read immediately; the remainder of the current record is skipped.

EXAMPLE:

READ(5,*,II,JJ,KK,LL,MM);

Input data: 987, 654, 321, 123/DSCENDING (CR) (LF)
456

Result: II contains 987
JJ contains 654
KK contains 321
LL contains 123
MM contains 456

List Terminator

If a line terminates ((CR) (LF)) and a slash has not been encountered, the input operation terminates even though all list elements may not have been processed. The current values of remaining elements are unchanged.

INPUT/OUTPUT

EXAMPLES:

```
READ(5,*,A,B,C,J,X,Y,Z);
```

Input Data:

A=7.987 B=5E2 C=4.6859E-3 (CR) (LF)

J=3456 (CR) (LF)

Result: A contains 7.987
B contains 5E2
C contains 4.6859E-3
J, X, Y, Z are unchanged.

Comments

All characters appearing between a pair of quotation marks in the same line are considered to be comments and are ignored by the compiler.

EXAMPLES:

"6.7321" is a comment and ignored.

6.7321 is a real number.

READ AND WRITE STATEMENTS

READ and WRITE statements have the following form:

```
READ (unit, format identifier, input list)
```

```
WRITE (unit, format identifier, output list)
```

The unit is an arithmetic expression with an integer value, which designates an I/O device. The format is the name of some declared format. The absence of the format identifier implies binary I/O. An asterisk (*) instead of a format identifier in a READ statement specifies free field input. The absence of a list means that no variables are to be input or output, but a format specification alone (e.g., nH) may produce input or output.

INPUT/OUTPUT

The input list or output list is a list of elements.

EXAMPLES:

```
READ (5, *, X, Y, Z, N, FOR I ← 1 TO N DO A [I]);  
WRITE (4, X, Y, X ↑ 2, Y ↑ 2, TRIG);
```

In-Line FORMAT Declarations

The information normally contained in a separate FORMAT declaration may be incorporated into a READ or WRITE statement by using the crosshatch (#) as follows:

```
FORMAT F (8I6,3F7.2);  
READ (1,F,A,B,C)  
may be written  
READ (1,#(8I6,3F7.2),A,B,C)
```

MAGNETIC TAPE STATEMENTS

Statements are available in HP ALGOL to handle HP magnetic tape operations:

```
ENDFILE unit  
REWIND unit  
SPACE unit  
BACKSPACE unit  
UNLOAD unit
```

INPUT/OUTPUT

UNIT-NUMBER

The integer specified for an input/output unit is a number that represents a unit assignment. The physical device referenced depends on tables established within the control system.

The standard unit numbers vary in meaning according to the operating system. Consult the appropriate system manual.

CARRIAGE CONTROL

PURPOSE: To indicate the line spacing used when printing an output record on a line printer. (The device driver interprets the first character in the data record as a carriage control character.)

FORMAT:

^ (blank)	}	as the first character in the record
0		
1		
*		
any other character	)	

^ = single space (print on every line).
0 = double space (print on every other line).
1 = eject page
* = suppress spacing (overprint current line).
any other character = single space (print on every line).

EXAMPLES:

<u>Format Specification</u>	<u>Result</u>
FORMAT F1("_PRINT ON EVERY LINE");	PRINT ON EVERY LINE
FORMAT F2("0PRINT ON EVERY OTHER LINE");	PRINT ON EVERY OTHER LINE
FORMAT F3("1");	(a page is ejected)
FORMAT F4("*PRINT ON CURRENT LINE");	(an overprint of current line)
FORMAT F5("PRINT ON EVERY LINE");	RINT ON EVERY LINE
FORMAT F6(1H1,2X,E16.8,2X,I5);	(A page is ejected, and a floating point number and an integer are then printed.)

SECTION IV

USING THE HP ALGOL COMPILER

CONTROL STATEMENT

The first record of any HP ALGOL program must be a control statement. The word HPAL is mandatory. Any combination of the following symbols may appear next, separated by commas:

- L: produce source program listing
- A: produce object code listing
- B: produce object tape
- P: a procedure only is to be compiled
- S: sense switch control (SIO environment only)

If no symbols are specified, the program will run but will not produce any output other than diagnostic messages. A program name in quotes (the NAM record name which must be a legitimate identifier with no blanks) is required and appears before the control symbols for SIO and after the symbols for DOS/RTE.

EXAMPLES:

HPAL,B,P, "INVRT" (DOS/RTE environment)
HPAL,"INVRT",B,P (SIO environment)

For SIO only, if the character "S" is included in the control statement, the B, L, and A options are read by the compiler from the switch register. The switches corresponding to the options:

<u>Switch (up)</u>	<u>Control Statement Equivalent</u>
15	B - produce object tape
14	L - produce source listing
13	A - produce object listing

The switches are read at the beginning of each line so that any option may be "turned off" part way through compilation. *Note that the P option, if used, must still be placed in the control statement.*

USING THE HP ALGOL COMPILER

OPERATING PROCEDURES

Operating procedures are found in the *SOFTWARE OPERATING PROCEDURES* (SIO Subsystem Module, 5951-1390) or the appropriate operating system manuals.

SECTION V

FORTRAN AND ASSEMBLER SUBROUTINES

An ALGOL program can refer to procedures that have been prepared using FORTRAN or assembly language; these subroutines are declared CODE procedures. (See Section II.) Object programs generated by the ALGOL or the FORTRAN compiler, or by the Assembler may then be linked by the Relocating Loader when the programs are loaded.

CALLING FORTRAN SUBROUTINES FROM ALGOL

A FORTRAN subroutine is called from an ALGOL program by declaring it a CODE procedure. Since FORTRAN and ALGOL are not fully compatible, the following rule should be obeyed:

If a parameter is an array, the formal parameter must be specified as REAL or INTEGER (depending on the type of array) rather than as ARRAY or INTEGER ARRAY. The actual parameter should be the address of the first element of the array.

EXAMPLE:

```
PROCEDURE F(A); REAL A; CODE;  
ARRAY A [1:10];  
F(A[1])
```

The following is wrong:

```
PROCEDURE F(A); ARRAY A; CODE;  
F(A);
```

CALLING ALGOL SUBROUTINES FROM FORTRAN

An ALGOL procedure can be called from FORTRAN if it is compiled with the P option in the control statement. Arrays can not be passed as parameters.

CALLING ALGOL PROCEDURES FROM ASSEMBLY LANGUAGE

The calling sequence to an ALGOL procedure from assembly Language is:

```

EXT    proc
JSB    proc
DEF    <return address>
DEF    par1
DEF    par2
      ⋮
DEF    parn
<return address>...
```

The ALGOL call would be:

```
proc (par1, par2, ..., parn)
```

CALLING ASSEMBLY LANGUAGE PROCEDURES FROM ALGOL

Assembly language procedures called from ALGOL either obtain parameters from the calling sequence given for calling from assembly language or use the standard ALGOL entry section:

```

ENT    P
EXT    .PRAM
P      NOP
JSB    .PRAM
< code words >
< parameters >
      ...
JMP    P, I
```

The code words have the following format:

First Code word:	bits 15 - 10 = number of parameters
	bits 9 - 8 = bit pair for first parameter
	bits 7 - 6 = bit pair for 2 nd parameter
	...
	bits 1 - 0 = bit pair for 5 th parameter

FORTRAN AND ASSEMBLER SUBROUTINES

Second Code Word: bits 15 - 14 = bit pair for 6th parameter
 ...
 bits 1 - 0 = bit pair for 13th parameter
 etc.

A bit pair for a parameter is as follows: the first bit is 1 if the parameter is called by value, 0 if it is called by name. The second bit has no significance for name parameters; while for value parameters, it is 1 for real variables and 0 for integer variables.

The locations which follow the code words must contain exactly enough words for the parameters. There are two words for each REAL, VALUE parameter, and one word for every other parameter.

EXAMPLE:

```
PROCEDURE P(A, B, X, Y);
  VALUE A, B;  INTEGER A, X:  REAL  B, Y;
P  NOP
   JSB  .PRAM
   OCT  11300

A  BSS  1      VALUE INTEGER
B  BSS  2      VALUE REAL
X  BSS  1      NAME
Y  BSS  1      NAME
START.....
```

All references to A and B should be direct, since they are values. All references to X and Y should be indirect, since they are addresses.

APPENDIX A

ALGOL ERROR MESSAGES

COMPILER MESSAGES

Errors detected in the source program are indicated by a numeric error code. In addition, an arrow (↑) is printed below the symbol which caused the error. Compiler error codes and a description of the condition causing the error follow:

<u>Error Code</u>	<u>Description</u>
1	More than two characters used in an ASCII constant.
2	@ not followed by an octal digit.
3	Octal constant greater than 177777.
4	Two decimal points in one number.
5	Non-integer following apostrophe.
6	Label declared but not defined in program.
7	Number required but not present.
8	Missing END
10	Undefined identifier.
11	Illegal symbol.
12	Procedure designator must be followed by left parenthesis.
13	Parameter types disagree.
14	Name parameter may not be an expression.
15	Parameter must be followed by a comma or right parenthesis.
16	Too many parameters.
17	Too few parameters.
18	Array variable not followed by a left bracket.
19	Subscript must be followed by a comma or right bracket.
20	Missing THEN.
21	Missing ELSE.
22	Illegal assignment.
23	Missing right parenthesis.
24	Proper procedure not legal in arithmetic expression.
25	Primary may not begin with this type quantity.

ALGOL ERROR MESSAGES

<u>Error Code</u>	<u>Description</u>
26	Too many subscripts.
27	Too few subscripts.
28	Variable required.
40	Too many external symbols.
41	Declarative following statement.
42	No parameters declared after left parenthesis.
43	REAL, INTEGER, or BOOLEAN illegal with this declaration.
44	Doubly defined identifier or reserved word found.
45	Illegal symbol in declaration.
46	Statement started with illegal symbol.
47	Label not followed by colon.
48	Label is previously defined.
49	Semicolon expected as terminator.
50	Left arrow or := expected in SWITCH declaration.
51	Label entry expected in SWITCH declaration.
52	Real number assigned to integer.
53	Constant expected following left arrow or :=
54	Left arrow or := expected in EQUATE declaration.
55	Left bracket expected in array declaration.
56	Integer expected in array dimension.
57	Colon expected in array dimension.
58	Upper bound less than lower bound in array.
59	Right bracket expected at end of array dimensions.
60	Too many values for array initialization.
61	Array size excessive (set to 2047).
62	Constant expected in array initialization.
63	Too many parameters for procedure.
64	Right parenthesis expected at end of procedure parameter list.
65	Procedure parameter descriptor missing.
66	VALUE parameter for procedure not in list.
67	Illegal TYPE found in procedure declaration.
68	Illegal description in procedure declaratives.
69	Identifier not listed as procedure parameter.
70	No type FOR variable in procedure parameter list.

ALGOL ERROR MESSAGES

<u>Error Code</u>	<u>Description</u>
71	Semicolon found in a format declaration.
72	Left parenthesis expected after I/O declaration name.
73	Right parenthesis expected after I/O name parameters.
74	Undefined label reference.
75	Switch identifier not followed by a left bracket.
76	Missing right bracket in switch designator.
77	THEN missing in IF statement.
78	DO missing in WHILE statement.
79	FOR variable must be of type INTEGER.
80	FOR variable must be followed by an assign symbol.
81	STEP symbol missing in FOR clause.
82	UNTIL symbol missing in FOR clause or DO statement.
83	DO symbol missing in FOR clause.
84	Parenthesis expected in READ/WRITE statement.
85	Comma expected in READ/WRITE statement.
86	Free field format (*) illegal with WRITE.
87	Unmatched [in I/O statement list.
88	Missing BEGIN in case statement.
89	Missing END in case statement.
100	Program must start with BEGIN, REAL, INTEGER or PROCEDURE.
999	Table areas overflowed.

RUN-TIME MESSAGES

Errors that occur during execution of the object program result in error messages at the console. Message format depends on the source of the error. Errors can result from equipment, format, or indexing errors, as well as errors occurring in a relocatable subroutine called by the program being run.

Run-time error messages are listed and described in Appendix B.

APPENDIX B

RUN-TIME ERROR MESSAGES

RELOCATABLE SUBROUTINE LIBRARY MESSAGES

During execution of programs which reference relocatable subroutines, error messages may be generated. For mathematical subroutines, the error message consists of the program name together with a two-digit subroutine identifier and a two-character error type indicator. The message is printed in the form:

program name nn error type

program name is the name of the user program through which the error was generated.

nn is a number in the range 02-14 (decimal) that identifies the subroutine which generated the error.

error type is OF for integer or floating point overflow, OR for out of range, or UN for floating point underflow.

Mathematical subroutine error codes follow:

<u>Error Code</u>	<u>Subroutine</u>	<u>Condition</u>
02-UN	ALOG, ALOGT	$x \leq 0$
02-UN	CLOG	$x = 0$
03-UN	DSQRT, SQRT	$x < 0$
04-UN	.RTOR	$x = 0, y \leq 0$ $x < 0, y \neq 0$
05-OR	CEXP	$\frac{1}{2} \left \frac{x}{\pi} + \frac{1}{2} \right > 2^{14}$
05-OR	CSNCS, SICOS	$\frac{1}{2} \left \frac{x}{\pi} + \frac{1}{2} \right > 2^{14}$
06-UN	.RTOI	$x = 0, i \leq 0$

RUN-TIME ERROR MESSAGES

<u>Error Code</u>	<u>Subroutine</u>	<u>Condition</u>
07-OF	CEXP	$x_1 \cdot \log_2 e \geq 124$
07-OF	CSNCS	$x_2 \cdot \log_2 e \geq 124$
07-OF	EXP	$x \cdot \log_2 e \geq 124$
07-OF	.RTOR	$ x \cdot \text{ALOG}(x) \geq 124$
08-OF	.ITOI	$i^j \geq 2^{23}$
08-UN	.ITOI	$i = 0, j \leq 0$
09-OR	TAN	$x > 2^{14}$
10-OF	DEXP	$e^x > (1 - 2^{-39})2^{127}$
10-OF	.DTOD, .DTOR, .RTOD	$x > (1 - 2^{-39})2^{127}$
11-UN	DLOG, DLOGT	$x \leq 0$
12-UN	.DIOI	$x = 0, i \leq 0$
13-UN	.DTOD, .DTOR, .RTOD	$x = 0, y \leq 0$ $x < 0, y \neq 0$
14-UN	.CTOI	$x = 0, i \leq 0$

INDEX? DIAGNOSTIC MESSAGE

The message INDEX? appears during execution whenever an attempt is made to access an array with an invalid or illegal index. For SIO, the computer halts after the message is printed. The A-register contains the address at which illegal reference was made. Press RUN to continue program execution from that point. For DOS/RTE, the address in violation is printed and execution continues.

RUN-TIME ERROR MESSAGES

OBJECTIVE PROGRAM MESSAGES

During execution of the object program, diagnostic messages are transmitted to the console. When an error is encountered, a message is printed and the computer halts. A code which further defines the error condition is contained in the A-register:

<u>Error Message</u>	<u>A-register</u>	<u>Explanation</u>	<u>Action</u>
*EQR	Unit Number	SIO environment only. Equipment error: end of input tape on tele- printer or tape reader; tape supply low on tape punch. B-register con- tains status word of equipment table entry.	Place next tape into the input device, or for tape punch, load new reel of tape. Press RUN.
*FMT	000001	FORMAT error: a) w or d field does not contain proper digits. b) No decimal point after w field. c) w-d 4 for E specification.	Irrecoverable error; program must be recompiled.
*FMT	000002	a) FORMAT specifications are nested more than one level deep. b) FORMAT statement con- tains more right parentheses than left parentheses.	Irrecoverable error; program must be recompiled.

RUN-TIME ERROR MESSAGES

<u>Error Message</u>	<u>A-register</u>	<u>Explanation</u>	<u>Action</u>
*FMT	000003	a) FORMAT statement contains illegal character. b) FORMAT repetition factor of zero.	Irrecoverable error; program must be recompiled.
*FMT	000004	Illegal character in fixed field input item or number not right- justified in field.	Verify data.
*FMT	000005	A number has an illegal form (For example, two E characters, two decimal points, two signs, and so forth.)	Verify data.

APPENDIX C

EXAMPLES

EXAMPLE 1

Line 008 declares an external procedure, "PMDIV";
line 023 calls "PMDIV". See EXAMPLE 2.

PAGE 001

```
001 <control statement>
002 02000 BEGIN COMMENT THIS TEST INCLUDES INTEGER ARRAYS,
003 02003      INTEGER DECLARATIONS,MOD OPERATORS,LABELS
004 02003      FOR STATEMENTS AND IF STATEMENTS;
005 02003 INTEGER ARRAY PRIME[1:500], DIVISOR[1:50];
006 03052 INTEGER I,N,D,K;
007 03052 INTEGER L,H;
008 03052 PROCEDURE PMDIV (A,N,B,L,H);
009 03053 VALUE H;  INTEGER N,L,H;  INTEGER ARRAY A,B;
010 03053 CODE;
011 03052 LABEL L2,RPEAT;
012 03054 OUTPUT TABLE(FOR K←1  TO L-1 DO[DIVISOR[K]]);
013 03101 PRIME[1]←2;
014 03110 I←2;
015 03112 FOR N←3 TO 1000 DO
016 03123 BEGIN FOR D←2 TO N-1 DO
017 03135 IF N MOD D=0 THEN GO TO L2;
018 03153   PRIME[I]←N;
019 03162   I←I+1;
020 03165 L2: END;
021 03171 RPEAT:  WRITE(2, #("ENTER H"));
022 03206 READ(1,*,H);
023 03215 PMDIV(PRIME,I,DIVISOR,L,H);
024 03224 WRITE(2, #("H=",I6, //, "DIVISORS", /, 10I6), H, TABLE);
025 03255 PAUSE;
026 03256 GO TO RPEAT;
027 03257 END$
```

PROGRAM= 001260 BASE PAGE= 000027 ERRORS=000

EXAMPLES

EXAMPLE 2

PAGE 001

```
001 <control statement>
002 02000 COMMENT THIS PROGRAM IS AN EXTERNAL PROGRAM USED
003 02000      TO COMPUTE THE PRIME DIVISORS OF NUMBERS BETWEEN
004 02000      1 AND 200;
005 02000 PROCEDURE PMDIV (A,N,B,L,H);
006 02001 VALUE H; INTEGER N,L,H; INTEGER ARRAY A,B;
007 02001 BEGIN INTEGER J,K; LABEL L1,L2;
008 02013 L←1;
009 02015 K←1;
010 02017 L1: FOR J←1 TO N-1 DO
011 02031 BEGIN IF H\K=1 THEN GO TO L2;
012 02043 IF H\K MOD A[J]=0 THEN
013 02070 BEGIN B[L]←A[J];
014 02102 K←A[J]*K;
015 02111 L←L+1;
016 02114 END;
017 02114 END;
018 02120 GO TO L1;
019 02121 L2: END;
```

PROGRAM= 000122 BASE PAGE= 000010 ERRORS=000

EXAMPLES

EXAMPLE 3

PAGE 001

```

001 <control statement>
002 02000 BEGIN COMMENT THIS TEST INCLUDES INTEGER ARRAYS,
003 02003      INTEGER DECLARATIONS,MOD OPERATORS,LABELS
004 02003      FOR STATEMENTS AND IF STATEMENTS;
005 02003 INTEGER ARRAY PRIME[1:100];
006 02150 INTEGER I,N,D,K;
007 02150 LABEL L2;
008 02151 OUTPUT LISTING(FOR K←1 TO I-1 DO [PRIME[K]]);
009 02176   PRIME[1]←2;
010 02205   I←2;
011 02207 FOR N←3 TO 200 DO
012 02220 BEGIN FOR D←2 TO N-1 DO
013 02232 IF N MOD D=0 THEN GO TO L2;
014 02250   PRIME[I]←N;
015 02257   I←I+1;
016 02262 L2: END;
017 02266   WRITE (2, #(20X,"PRIMES",//10(I3,3X)),LISTING);
018 02314 END$

```

PROGRAM= 000315 BASE PAGE= 000021 ERRORS=000

Result of running "TESTE", above.

PRIMES

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199				

STOP
*NEXT?

EXAMPLES

EXAMPLE 4

PAGE 001

```

001 <control statement>
002 02000 BEGIN COMMENT THIS TEST TESTS MAG-TAPE COMMANDS;
003 02003 PROCEDURE PTAPE(X,Y,Z);
004 02005 VALUE X,Y,Z; INTEGER X,Y,Z; CODE;
005 02003 INTEGER ARRAY FI[1:3]←"FI","LE"," #";
006 02007 INTEGER ARRAY RC[1:4]←"RE","CO","RD"," #";
007 02013 INTEGER ARRAY BUFUR[1:9];
008 02024 INTEGER X,Y,Z,I,J,K,L,M,A,N;
009 02024 FORMAT F1(21HUNIT NO. OF MAG-TAPE?),F0("F",3A2,I2,10X,4A2,I2);
010 02055 FORMAT F2("FFILES AND RECORDS IN ORDER AS ON MAG-TAPE");
011 02104 FORMAT F3(" ");
012 02111 FORMAT F4("FFILES AND RECORDS IN REVERSE ORDER");
013 02135 FORMAT F5("FFIRST RECORD OF EACH FILE");
014 02154 OUTPUT FIREC(FI[1],FI[2],FI[3],I,RC[1],RC[2],RC[3],RC[4],J);
015 02225 OUTPUT BUF1(FOR M←1 TO 9 DO BUFUR[M]);
016 02251 INPUT BUFF(FOR K←1 TO 9 DO BUFUR[K]);
017 02277 LABEL BS,LN;
018 02301 WRITE(2,F1);
019 02307 READ(1,*,A);
020 02316 Y←2; Z←0;
021 02322 PTAPE(A,Y,Z);
022 02327 FOR I←1 TO 10 DO
023 02340 BEGIN FOR J←1 TO 10 DO
024 02351 BEGIN WRITE(A,F0,FIREC);
025 02360 END;
026 02364 ENDFILE A;
027 02367 END;
028 02373 REWIND A;
029 02376 PTAPE(A,Y,Z);
030 02403 WRITE(6,F2);
031 02411 FOR X←1 TO 3 DO WRITE(6,F3);
032 02434 FOR X←1 TO 10 DO
033 02445 BEGIN
034 02445 FOR L←1 TO 10 DO
035 02456 BEGIN READ(A,F0,BUFF);
036 02464 WRITE(6,F0,BUF1);
037 02473 END;
038 02477 SPACE A;
039 02502 END;
040 02506 FOR X←1 TO 9 DO WRITE(6,F3);
041 02531 WRITE(6,F4);
042 02537 FOR X←1 TO 3 DO WRITE(6,F3);
043 02562 N←0;
044 02564 BS: FOR I←1 TO 10 DO
045 02575 BEGIN BACKSPACE A;
046 02600 BACKSPACE A;
047 02603 READ(A,F0,BUFF);
048 02611 WRITE(6,F0,BUF1);
049 02620 END;

```

EXAMPLES

PAGE 002

```

050 02624      N←N+1;
051 02627      IF N<10 THEN BACKSPACE A ELSE GO TO LN;
052 02637      GO TO BS;
053 02640 LN:   BACKSPACE A;
054 02643      FOR X←1 TO 9 DO WRITE(6,F3);
055 02666      WRITE(6,F5);
056 02674      FOR X←1 TO 3 DO WRITE(6,F3);
057 02717      FOR I←1 TO 10 DO
058 02730      BEGIN READ(A,F0,BUFF);
059 02736          WRITE(6,F0,BUF1);
060 02745          FOR J←1 TO 10 DO SPACE A;
061 02765      END;
062 02771      END$

```

PROGRAM= 000772 BASE PAGE= 000051 ERRORS=000

Result of running "TESTX"

FILES AND RECORDS IN ORDER AS ON MAG-TAPE

FILE # 1	RECORD # 1
FILE # 1	RECORD # 2
FILE # 1	RECORD # 3
FILE # 1	RECORD # 4
FILE # 1	RECORD # 5
FILE # 1	RECORD # 6
FILE # 1	RECORD # 7
FILE # 1	RECORD # 8
FILE # 1	RECORD # 9
FILE # 1	RECORD #10
FILE # 2	RECORD # 1
FILE # 2	RECORD # 2
FILE # 2	RECORD # 3
FILE # 2	RECORD # 4
FILE # 2	RECORD # 5
FILE # 2	RECORD # 6
FILE # 2	RECORD # 7
FILE # 2	RECORD # 8
FILE # 2	RECORD # 9
FILE # 2	RECORD #10
FILE # 3	RECORD # 1
FILE # 3	RECORD # 2
FILE # 3	RECORD # 3
FILE # 3	RECORD # 4
FILE # 3	RECORD # 5
FILE # 3	RECORD # 6
FILE # 3	RECORD # 7
FILE # 3	RECORD # 8
FILE # 3	RECORD # 9
FILE # 3	RECORD #10

EXAMPLES

FILE # 4	RECORD # 1
FILE # 4	RECORD # 2
FILE # 4	RECORD # 3
FILE # 4	RECORD # 4
FILE # 4	RECORD # 5
FILE # 4	RECORD # 6
FILE # 4	RECORD # 7
FILE # 4	RECORD # 8
FILE # 4	RECORD # 9
FILE # 4	RECORD #10
FILE # 5	RECORD # 1
FILE # 5	RECORD # 2
FILE # 5	RECORD # 3
FILE # 5	RECORD # 4
FILE # 5	RECORD # 5
FILE # 5	RECORD # 6
FILE # 5	RECORD # 7
FILE # 5	RECORD # 8
FILE # 5	RECORD # 9
FILE # 5	RECORD #10
FILE # 6	RECORD # 1
FILE # 6	RECORD # 2
FILE # 6	RECORD # 3
FILE # 6	RECORD # 4
FILE # 6	RECORD # 5
FILE # 6	RECORD # 6
FILE # 6	RECORD # 7
FILE # 6	RECORD # 8
FILE # 6	RECORD # 9
FILE # 6	RECORD #10
FILE # 7	RECORD # 1
FILE # 7	RECORD # 2
FILE # 7	RECORD # 3
FILE # 7	RECORD # 4
FILE # 7	RECORD # 5
FILE # 7	RECORD # 6
FILE # 7	RECORD # 7
FILE # 7	RECORD # 8
FILE # 7	RECORD # 9
FILE # 7	RECORD #10
FILE # 8	RECORD # 1
FILE # 8	RECORD # 2
FILE # 8	RECORD # 3
FILE # 8	RECORD # 4
FILE # 8	RECORD # 5
FILE # 8	RECORD # 6
FILE # 8	RECORD # 7
FILE # 8	RECORD # 8
FILE # 8	RECORD # 9
FILE # 8	RECORD #10

EXAMPLES

FILE # 9	RECORD # 1
FILE # 9	RECORD # 2
FILE # 9	RECORD # 3
FILE # 9	RECORD # 4
FILE # 9	RECORD # 5
FILE # 9	RECORD # 6
FILE # 9	RECORD # 7
FILE # 9	RECORD # 8
FILE # 9	RECORD # 9
FILE # 9	RECORD #10
FILE #10	RECORD # 1
FILE #10	RECORD # 2
FILE #10	RECORD # 3
FILE #10	RECORD # 4
FILE #10	RECORD # 5
FILE #10	RECORD # 6
FILE #10	RECORD # 7
FILE #10	RECORD # 8
FILE #10	RECORD # 9
FILE #10	RECORD #10

FILES AND RECORDS IN REVERSE ORDER

FILE #10	RECORD #10
FILE #10	RECORD # 9
FILE #10	RECORD # 8
FILE #10	RECORD # 7
FILE #10	RECORD # 6
FILE #10	RECORD # 5
FILE #10	RECORD # 4
FILE #10	RECORD # 3
FILE #10	RECORD # 2
FILE #10	RECORD # 1
FILE # 9	RECORD #10
FILE # 9	RECORD # 9
FILE # 9	RECORD # 8
FILE # 9	RECORD # 7
FILE # 9	RECORD # 6
FILE # 9	RECORD # 5
FILE # 9	RECORD # 4
FILE # 9	RECORD # 3
FILE # 9	RECORD # 2
FILE # 9	RECORD # 1
FILE # 8	RECORD #10
FILE # 8	RECORD # 9
FILE # 8	RECORD # 8
FILE # 8	RECORD # 7
FILE # 8	RECORD # 6
FILE # 8	RECORD # 5
FILE # 8	RECORD # 4
FILE # 8	RECORD # 3
FILE # 8	RECORD # 2
FILE # 8	RECORD # 1

EXAMPLES

FILE # 7	RECORD #10
FILE # 7	RECORD # 9
FILE # 7	RECORD # 8
FILE # 7	RECORD # 7
FILE # 7	RECORD # 6
FILE # 7	RECORD # 5
FILE # 7	RECORD # 4
FILE # 7	RECORD # 3
FILE # 7	RECORD # 2
FILE # 7	RECORD # 1
FILE # 6	RECORD #10
FILE # 6	RECORD # 9
FILE # 6	RECORD # 8
FILE # 6	RECORD # 7
FILE # 6	RECORD # 6
FILE # 6	RECORD # 5
FILE # 6	RECORD # 4
FILE # 6	RECORD # 3
FILE # 6	RECORD # 2
FILE # 6	RECORD # 1
FILE # 5	RECORD #10
FILE # 5	RECORD # 9
FILE # 5	RECORD # 8
FILE # 5	RECORD # 7
FILE # 5	RECORD # 6
FILE # 5	RECORD # 5
FILE # 5	RECORD # 4
FILE # 5	RECORD # 3
FILE # 5	RECORD # 2
FILE # 5	RECORD # 1
FILE # 4	RECORD #10
FILE # 4	RECORD # 9
FILE # 4	RECORD # 8
FILE # 4	RECORD # 7
FILE # 4	RECORD # 6
FILE # 4	RECORD # 5
FILE # 4	RECORD # 4
FILE # 4	RECORD # 3
FILE # 4	RECORD # 2
FILE # 4	RECORD # 1
FILE # 3	RECORD #10
FILE # 3	RECORD # 9
FILE # 3	RECORD # 8
FILE # 3	RECORD # 7
FILE # 3	RECORD # 6
FILE # 3	RECORD # 5
FILE # 3	RECORD # 4
FILE # 3	RECORD # 3
FILE # 3	RECORD # 2
FILE # 3	RECORD # 1

EXAMPLES

FILE # 2	RECORD #10
FILE # 2	RECORD # 9
FILE # 2	RECORD # 8
FILE # 2	RECORD # 7
FILE # 2	RECORD # 6
FILE # 2	RECORD # 5
FILE # 2	RECORD # 4
FILE # 2	RECORD # 3
FILE # 2	RECORD # 2
FILE # 2	RECORD # 1
FILE # 1	RECORD #10
FILE # 1	RECORD # 9
FILE # 1	RECORD # 8
FILE # 1	RECORD # 7
FILE # 1	RECORD # 6
FILE # 1	RECORD # 5
FILE # 1	RECORD # 4
FILE # 1	RECORD # 3
FILE # 1	RECORD # 2
FILE # 1	RECORD # 1

FIRST RECORD OF EACH FILE

FILE # 1	RECORD # 1
FILE # 2	RECORD # 1
FILE # 3	RECORD # 1
FILE # 4	RECORD # 1
FILE # 5	RECORD # 1
FILE # 6	RECORD # 1
FILE # 7	RECORD # 1
FILE # 8	RECORD # 1
FILE # 9	RECORD # 1
FILE #10	RECORD # 1

INDEX

A

Arithmetic expression.....1-6
 ARRAY.....1-2
 Array declarations.....1-2
 ASCII
 Character set.....x
 Constants.....xii
 Assembler.....5-1
 Assignment statements.....1-10

B

Block.....1-5
 BOOLEAN.....1-2,1-14
 Boolean
 Array declaration.....1-2
 Constants.....xiii
 Expressions.....1-14
 Type declaration.....1-1
 Variables.....1-15

C

Call by Name/Value.....2-3
 Carriage control.....3-23
 CASE STATEMENT.....1-20
 CHARACTER SET.....x
 CODE.....5-1
 Code procedures.....2-6
 COMMENT.....xiii
 Compiler.....4-1
 Compiler errors.....A-1
 COMPOUND STATEMENTS.....1-17
 Conditional expressions.....1-8
 Conditional statements.....1-12
 Constants
 ASCII.....xii
 Boolean.....xiii
 Decimal.....xii
 Octal.....xiii
 Control statement.....4-1

D

Decimal Constants.....xii
 Declarations.....1-1
 Array.....1-2
 Equate.....1-4
 Format.....3-2
 Label.....1-3
 List.....3-1

 Procedure.....2-1
 Switch.....1-3
 Type.....1-1
 Designational expression.....1-12
 Diagnostic Messages.....A-1,B-1
 DO.....1-16

E

EQUATE.....1-4
 Error Messages.....A-1,B-1
 EXAMPLES.....C-1
 Expressions
 Arithmetic.....1-6
 Boolean.....1-14
 Conditional.....1-8
 Designational.....1-12

F

FOR statements.....1-18
 FORMAT declarations.....3-2
 Format Declarations, In-line..3-22
 Format Specifications.....3-2
 FORTRAN.....5-1
 Function Procedures.....2-5
 Fundamentals.....x

G

GO TO.....1-11

H

HARDWARE CONFIGURATION.....ix

I

Identifiers.....xi
 In-line Format Declarations...3-22
 INTEGER.....1-1
 Intrinsic procedures.....2-6

L

LABEL.....1-3,1-12
 Label declaration.....1-3
 Labels.....1-12
 Lists.....3-1

M

Magnetic Tape Statements.....3-22

INDEX

N

NAME.....2-3

O

Octal constants.....xiii

Operators.....1-6

P

PAUSE.....1-21

Procedures.....2-1

 Code.....2-6

 Function.....2-5

 Intrinsic.....2-6

Program listing.....C-1

PROCEDURE.....2-1,2-3

R

READ statement.....3-21

REAL.....1-1

RESERVED IDENTIFIER.....xi

RUN-TIME ERRORS.....B-1

S

Statements.....1-10

 Assignment.....1-10

 CASE.....1-20

 Compound.....1-17

 Conditional.....1-12

 DO.....1-16

FOR.....1-18

GO TO.....1-11

Magnetic tape.....3-22

PAUSE.....1-21

Procedure.....2-3

READ.....3-21

WHILE.....1-16

WRITE.....3-21

STEP.....1-18

Subscripts.....1-9

SWITCH.....1-3

T

Type.....1-1,1-6

U

Unit-number.....3-23

UNTIL.....1-16

V

VALUE.....2-3

Variables.....1-9

 Boolean.....1-15

 Left part.....1-10

 Simple.....1-9

 Subscripted.....1-9

W

WHILE.....1-16

WRITE.....3-21

READER COMMENT SHEET

02116-9072

Oct 1974

HP ALGOL

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate?

Is this manual complete?

Is this manual easy to read and use?

Other comments?

FROM:

Name _____

Company _____

Address _____

FOLD

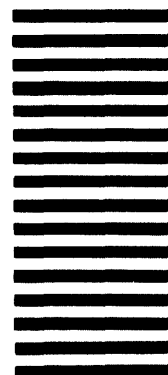
FOLD

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States Postage will be paid by

Manager, Technical Publications
Hewlett-Packard
Data Systems Development Division
11000 Wolfe Road
Cupertino, California 95014

FIRST CLASS
PERMIT NO. 141
CUPERTINO
CALIFORNIA



FOLD

FOLD



MANUAL PART NO. 02116-9072
MICROFICHE PART NO. 02116-91785

Printed: OCT 1974
Printed in U.S.A.